Derivative Free Methods of Optimization: Trust Region Method and Direct Search Method

A Thesis
Submitted by

CHINMOY DEY

Under the Guidance
Of

Prof. Debajyoti Choudhuri

DEPARTMENT OF MATHEMATICS
NATIONAL INSTITUTE OF TECHNOLOGY, ROURKELA
ROURKELA-769008

JUNE 2015

DECLARATION

I hereby declare that the topic "Derivative Free Methods of Optimization: Trust Region Method and Direct Search Method" for completion for my Master degree, under my best knowledge and honest of beliefs, has not been submitted in any other University or Institute for awards of any other form of degree or diploma.

Date:

Place:

CHINMOY DEY

Roll no: 410MA5048

Department of Mathematics

NIT Rourkela

CERTIFICATE

This is to certify that the project report entitled "Derivative Free Methods of Optimization: Trust Region Method and Direct Search Method" submitted by Chinmoy Dey to National Institute of Technology Rourkela, Orissa for his partial completion of requirements for his degree in and as Master of Science in Mathematics is an authentic and bona fide record of review of work and effort put forward by him under my guidance and supervision. The contents concerned to this project, in parts or in full, have never been submitted to any other University or Institute for any award of any form of diploma or degree.

June, 2015

(Prof. Debajyoti Choudhuri)
Associate Professor
Department of Mathematics
NIT Rourkela- 769008

ACKNOWLEDGEMENTS

I warmly express my humble feeling of gratitude and respect to Prof. Suvendu Ranjan Pattanaik, Department of Mathematics, NIT Rourkela, Odisha, for his consistent and expertise direction, suggestions during the continuation of my project and overall preparation of the project.

I would also like to thank the faculty members of Department of Mathematics, NIT Rourkela, who have always inspired me to be hard working and helped me to grasp new concepts during our course duration at NIT Rourkela. I also feel obligated to thank Himani Garg, 413MA2060, for her regular and honest support.

I also feel blessed about my family members for their unconditional support. They have inspired me in every scenario.

I will be indebted to all my juniors and friends for their constant co-operation during my project work.

Rourkela, 769008
June 2015

(Chinmoy Dey)

ABSTRACT

In this report, I applied non derivative methods as optimization techniques to find out the maximum or minimum value for different kinds of functions through computational methods. In the process, I have analyzed functions that are not differentiable or have high complexity with optimization output and tested the effectiveness of the methods during such cases. Also, I have analyzed the computational advantages of these methods over prevailing methods of optimization and its extent.

Table of Contents

Introduction

In my thesis, the two derivative free methods, namely Trust Free Method and Direct Search Method are being used to optimize functions. Their algorithm deals with matrices depending on the parameters of the function to iterate the process until an optimized value is calculated, depending on the tolerance level. In the next step, I have shown the enhanced effectiveness of this method in computation and otherwise, as it can be used for functions that are not differentiable or whose optimized values are hard to determine. Finally, I have analyzed the complexity of this method in comparison to other prevailing methods to show the added benefits of this method over normal ones.

CHAPTER 1

1. Direct Search Method

Accordingthe direct searchmethod,thealgorithmis made to choosea direction p_k andsearching is being donealongthat directionfromtheongoingiteratex_k foranewiteratewithalowervalue of the function.Thedistance moving along p_k can be conveniently found by solving the following one-dimensional minimization problem approximately to find a step length α:

$$\min_{\alpha>0} f(x_k + \alpha p_k)$$

By solving it exactly, we can derive the maximum requirement from the direction p_k, but an exact minimization requires high cost and is not required. Rather, direct search algorithm can produce a limited number of probationary step lengths until it finds one that roughlyapproaches theminimum.Atthisnewpoint,anewsearchdirectionandsteplengtharecalculated, and the process iterates.

2. Trust Region Method

In the second strategy of algorithm, known as the trust region, the datacongregated about f is used to paradigma model function m_k whose performance near the ongoing point x_k is comparable to that of the actual objective function f. Since the model m_k may not be a good estimate of f when x is far fromx_k, we limit the search for a minimizer of m_k to some limited regionaroundx_k. In other words, we find the candidate step p approximately by solving the following sub-problem:

$$\min_{p} m_k(x_k + p)$$

where x_k+p lies inside the trust region.

3. Algorithm:

ALGORITHM FOR TRUST REGION METHOD

Step 1:
Trial step computation: “Method = Trust Region”, compute S_k as one solution.

Step 2:
Compute $\rho(S_k) = \rho_q(S_k)$

If $\rho(S_k) \geq \eta_1$, let $x_{k+1} = x_k + S_k$

Else, $x_{k+1} = x_k$

Step 3:
Regulation Parameter Update

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty), if\ \rho(Sk) \geq \eta_2 (\text{Very successful}) \\ [\gamma_2 \Delta_k,\ \Delta_k],\ \text{if}\ \rho(Sk) \in [\eta_1, \eta_2) (\text{Successful}) \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k], \text{otherwise} \quad (\text{Unsuccessful}) \end{cases}$$

ALGORITHM FOR CO-ORDINATE DIRECT SEARCH METHOD

Initialization: Choose x_o and $\alpha_o > 0$

For k = 0, 1, 2…

Step 1: Poll Step

Order the poll set $\rho_k = \{ x_k + \alpha_k d : d \in D+\}$

Initiate evaluating f at the poll points subsequentto the order determined. If a poll point $x_k + \alpha_k d_k$ is found such that $f(x_k + \alpha_k d_k) < f(x_k)$, then stop polling. Set $x_{k+1} = x_k + \alpha_k d_k$, and declare the iteration and the poll step successful. Otherwise, declare the iteration unsuccessful and set $x_{k+1} = x_k$.

Step 2: Parameter update
If the iteration was successful, set $\alpha_{k+1} = \alpha_k$. Otherwise, set $\alpha_{k+1} = \alpha_k/2$.

ALGORITHM FOR DIRECTIONAL DIRECT SEARCH METHOD

Step1: Search Step
Try to compute a point with $f(x) < f(x_k)$ by evaluating the function f at a finite number of points. If such a point is found, then set $x_{k+1} = x$, declare the iteration and search step successful and the poll systems.

Step 2: Poll Step
Choose a positive basis D_k from the set D. Order of poll set $\rho_k = \{x_k + \alpha_k d: d \epsilon D_k\}$. Start evaluating f at all poll points following the chosen order. If a poll point $x_k + \alpha_k d_k$ is found such that $f(x_k + \alpha_k d_k) < f(x_k)$, then stop polling, set $x_{k+1} = x_k + \alpha_k d_k$ and declare

the iteration and poll step successful. Otherwise, announce the iteration unsuccessful and set $x_{k+1}= x_k$.

Step 3: Mesh Parameter Update

If the iteration is successful, maintain or increase the step size parameter. Otherwise, decrease the step size parameter.

CHAPTER 2

CODING, OUTPUT AND GRAPHS

4. Case 1: Function without a minimum

f(x) = e^x

Coding:

```
#include<iostream>
#include<math.h>
#include<cstdlib>
using namespace std;

float f(float x[], int size)
{
return (exp(x[0]));
}

int main()
{
int e,r,l,w;
cout<<"Enter  number of independent variables:";
cin>>w;

float x[w],y[w],del_tol=1e-19,del_not;
int D[2*w][w],C[w][w],B[2*w][w];
cout<<"Enter  initial assumption:";
for(int e=0;e<w;e++)
{
cin>>x[e];
}
```

```
cout<<"Enter the scale control parameter(del-not):";
cin>>del_not;

for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
{
if(e==r)
D[e][r]=1;
else
if(e>=w&&e==r+w)
D[e][r]=-1;
else
D[e][r]=0;
}
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
cout<<D[e][r]<<"  ";
cout<<"\n";
}
cout<<"\n Enter  matrix elements:";
for(e=0;e<w;e++)
{
for(r=0;r<w;r++)
cin>>C[e][r];
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
{
B[e][r]=0;
for(l=0;l<w;l++)
```

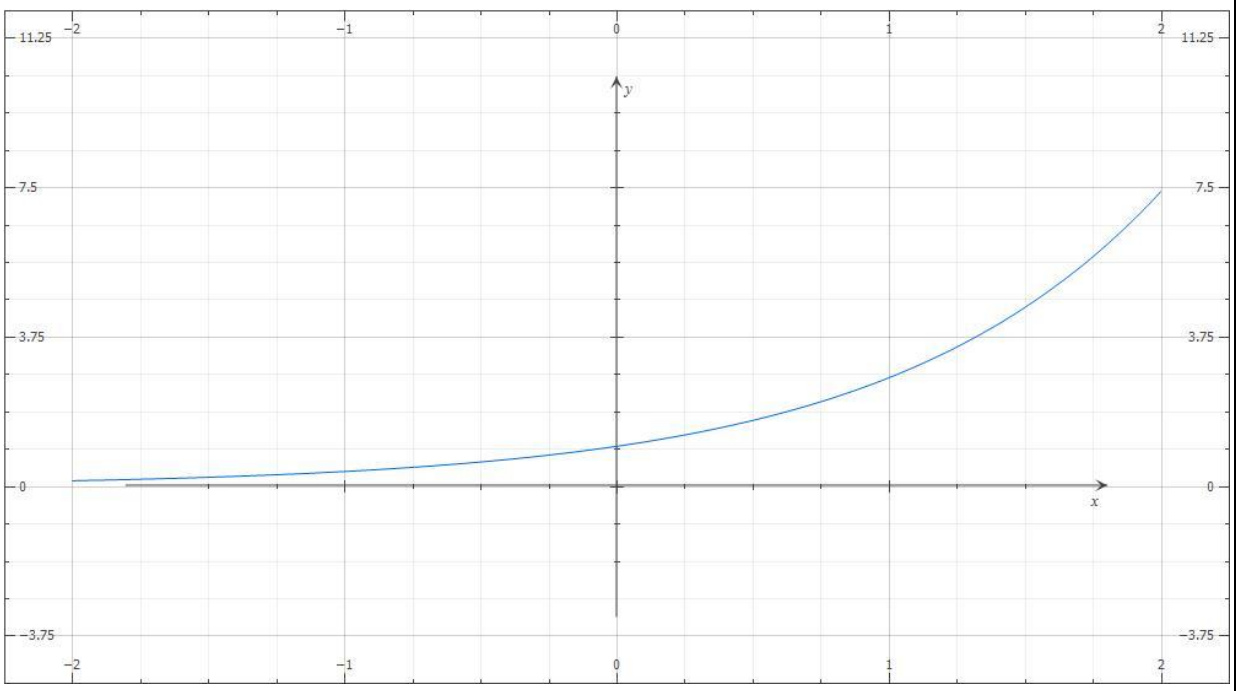
```
B[e][r]=B[e][r]+(D[e][l]*C[l][r]);
}
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
cout<<B[e][r]<<" ";
cout<<"\n";
}


if(del_not<del_tol)
cout<<"Enter scale parameter which is larger than tolerance:";
else
{
while(del_not>del_tol)
{
int flag=0;
for(int e=0;e<2*w;e++)
{
for(int r=0;r<w;r++)

y[r]=x[r]+B[e][r]*del_not;
if(f(y,w)<f(x,w)-pow(del_not,2))
{
for(int r=0;r<w;r++)
x[e]=y[r];
del_not=2*del_not;
flag=1;
break;
}



}
```

```
if(flag==0)
del_not/=2;
}
}

cout<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<"\t"<<x[3]<<endl;
cout<<"The minimum value is "<<f(x,d)<<endl;
cout<<"The step length value is "<<del_not<<endl;
return 0;
}
```

OUTPUT

```
Enter number of independent variables:  1
Enter initial assumption: 1
Enter scale control parameter <del-not>:0.1
Enter the matrix element: 1
1
-1
The minimum value is 6.664e-007
The step length value is 8.67362e-20
Compilation time: 6.73 sec
```

Graph:



5. Case 2: Function not differentiable at the minimum point

f(x) = |*x*|

Coding:

```
#include<iostream>
#include<math.h>
#include<cstdlib>
using namespace std;

float f(float x[], int size)
{
return (fabs(x[0]));
}

int main()
{
int e,r,l,w;
cout<<"Enter  number of independent variables:";
```

```
cin>>w;

float x[w],y[w],del_tol=1e-19,del_not;
int D[2*w][w],C[w][w],B[2*w][w];
cout<<"Enter  initial assumption:";
for(int e=0;e<w;e++)
{
cin>>x[e];
}
cout<<"Enter the scale control parameter(del-not):";
cin>>del_not;

for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
{
if(e==r)
D[e][r]=1;
else
if(e>=w&&e==r+w)
D[e][r]=-1;
else
D[e][r]=0;
}
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
cout<<D[e][r]<<"  ";
cout<<"\n";
}
cout<<"\n Enter  matrix elements:";
for(e=0;e<w;e++)
{
```

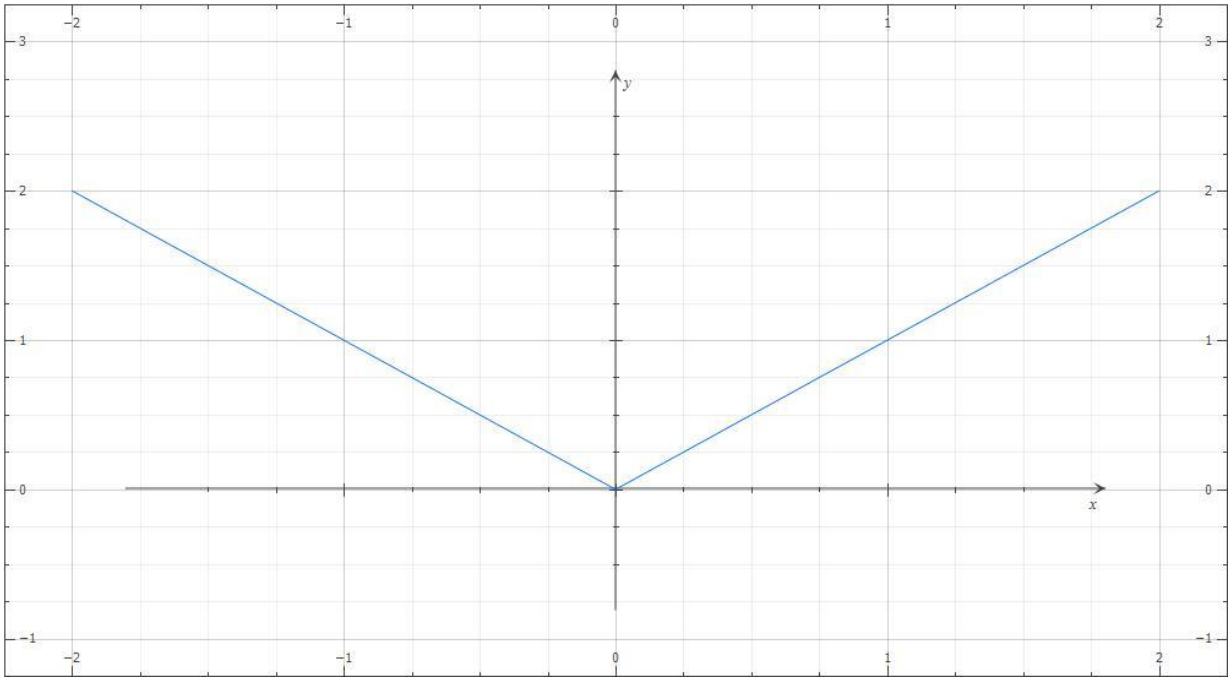
```
for(r=0;r<w;r++)
cin>>C[e][r];
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
{
B[e][r]=0;
for(l=0;l<w;l++)
B[e][r]=B[e][r]+(D[e][l]*C[l][r]);
}
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
cout<<B[e][r]<<" ";
cout<<"\n";
}


if(del_not<del_tol)
cout<<"Enter scale parameter which is larger than tolerance:";
else
{
while(del_not>del_tol)
{
int flag=0;
for(int e=0;e<2*w;e++)
{
for(int r=0;r<w;r++)

y[r]=x[r]+B[e][r]*del_not;
if(f(y,w)<f(x,w)-pow(del_not,2))
{
```

```
for(int r=0;r<w;r++)
x[e]=y[r];
del_not=2*del_not;
flag=1;
break;
}



}
if(flag==0)
del_not/=2;
}
}

cout<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<"\t"<<x[3]<<endl;
cout<<"The minimum value is "<<f(x,d)<<endl;
cout<<"The step length value is "<<del_not<<endl;
return 0;
}
```

OUTPUT

```
Enter number of independent variables:  1
Enter initial assumption: 1
Enter the scale control parameter <del-not>: .01
Entermatrix element : 1
1
-1
The minimum value is 3.62925e-23
The step length value is 6.93862e-20
Compilation time: 2.17 sec
```

Graph:



6. Case 3: Function not defined at a certain point

f(x) = x$sin\frac{1}{x}$

Coding:

```
#include<iostream>
#include<math.h>
#include<cstdlib>
using namespace std;

float f(float x[], int size)
```

```
{
return(x[0]*sin((1/x[0])));
}

int main()
{
int e,r,l,w;
cout<<"Enter  number of independent variables:";
cin>>w;

float x[w],y[w],del_tol=1e-19,del_not;
int D[2*w][w],C[w][w],B[2*w][w];
cout<<"Enter  initial assumption:";
for(int e=0;e<w;e++)
{
cin>>x[e];
}
cout<<"Enter the scale control parameter(del-not):";
cin>>del_not;

for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
{
if(e==r)
D[e][r]=1;
else
if(e>=w&&e==r+w)
D[e][r]=-1;
else
D[e][r]=0;
}
}
for(e=0;e<2*w;e++)
```

```
{
for(r=0;r<w;r++)
cout<<D[e][r]<<"  ";
cout<<"\n";
}
cout<<"\n Enter  matrix elements:";
for(e=0;e<w;e++)
{
for(r=0;r<w;r++)
cin>>C[e][r];
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
{
B[e][r]=0;
for(l=0;l<w;l++)
B[e][r]=B[e][r]+(D[e][l]*C[l][r]);
}
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
cout<<B[e][r]<<" ";
cout<<"\n";
}


if(del_not<del_tol)
cout<<"Enter scale parameter which is larger than tolerance:";
else
{
while(del_not>del_tol)
{
```

```
int flag=0;
for(int e=0;e<2*w;e++)
{
for(int r=0;r<w;r++)

y[r]=x[r]+B[e][r]*del_not;
if(f(y,w)<f(x,w)-pow(del_not,2))
{
for(int r=0;r<w;r++)
x[e]=y[r];
del_not=2*del_not;
flag=1;
break;
}



}
if(flag==0)
del_not/=2;
}
}

cout<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<"\t"<<x[3]<<endl;
cout<<"The minimum value is "<<f(x,d)<<endl;
cout<<"The step length value is "<<del_not<<endl;
return 0;
}
```

OUTPUT

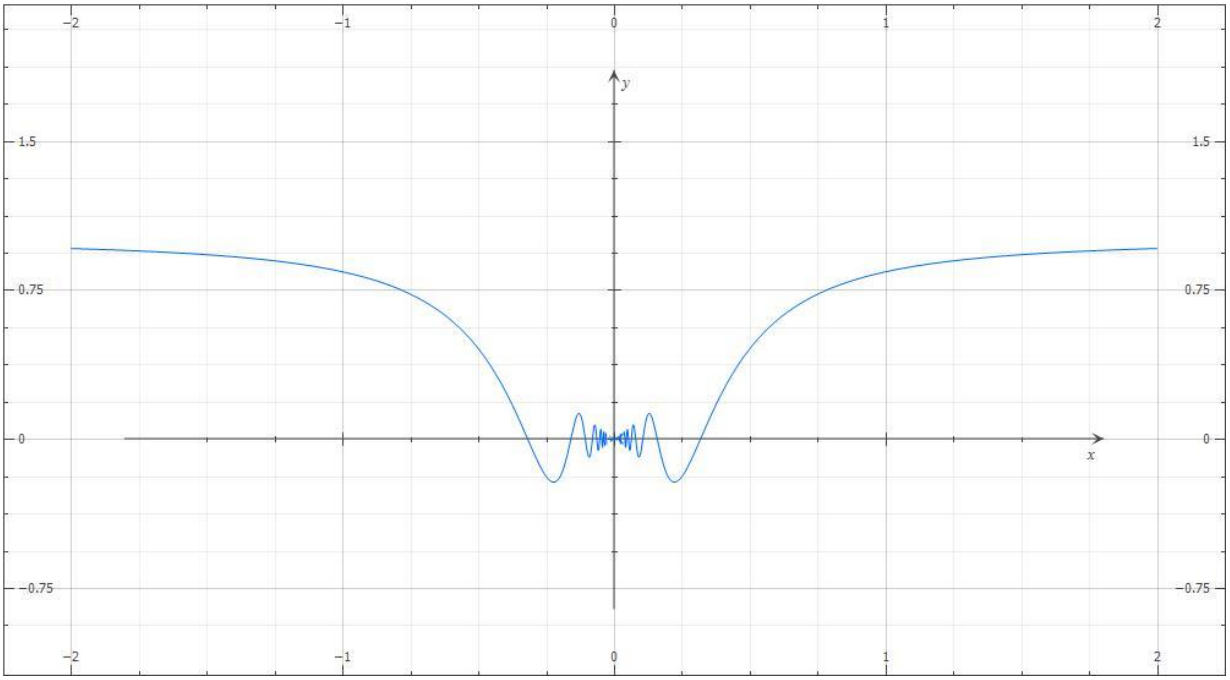
```
Enter number of independent variables:  1
Enter initial assumption: .1
Enter scale control parameter <del-not>: .01
Enter matrix elements: 1
1
-1
```

The minimum value is -0.217913252

The step length value is 6.93862e-20

Compilation time: 2.54sec

Graph:



7. Case 4: Combined cases

f(x) = $\frac{\sin x}{x}$ - e^{-x} cos x

Coding:

```
#include<iostream>
        #include<math.h>
        #include<cstdlib>
        using namespace std;

        float f(float x[], int size)
        {
          return(((sin(x[0]))/x[0])-exp(-x[0]*cos(x[0])));

        }

        int main()
        {
        int e,r,l,w;
        cout<<"Enter  number of independent variables:";
        cin>>w;

        float x[w],y[w],del_tol=1e-19,del_not;
        int D[2*w][w],C[w][w],B[2*w][w];
        cout<<"Enter  initial assumption:";
        for(int e=0;e<w;e++)
        {
        cin>>x[e];
        }
        cout<<"Enter the scale control parameter(del-not):";
        cin>>del_not;

        for(e=0;e<2*w;e++)
```

```
{
for(r=0;r<w;r++)
{
if(e==r)
D[e][r]=1;
else
if(e>=w&&e==r+w)
D[e][r]=-1;
else
D[e][r]=0;
}
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
cout<<D[e][r]<<"  ";
cout<<"\n";
}
cout<<"\n Enter  matrix elements:";
for(e=0;e<w;e++)
{
for(r=0;r<w;r++)
cin>>C[e][r];
}
for(e=0;e<2*w;e++)
{
for(r=0;r<w;r++)
{
B[e][r]=0;
for(l=0;l<w;l++)
B[e][r]=B[e][r]+(D[e][l]*C[l][r]);
}
}
for(e=0;e<2*w;e++)
```

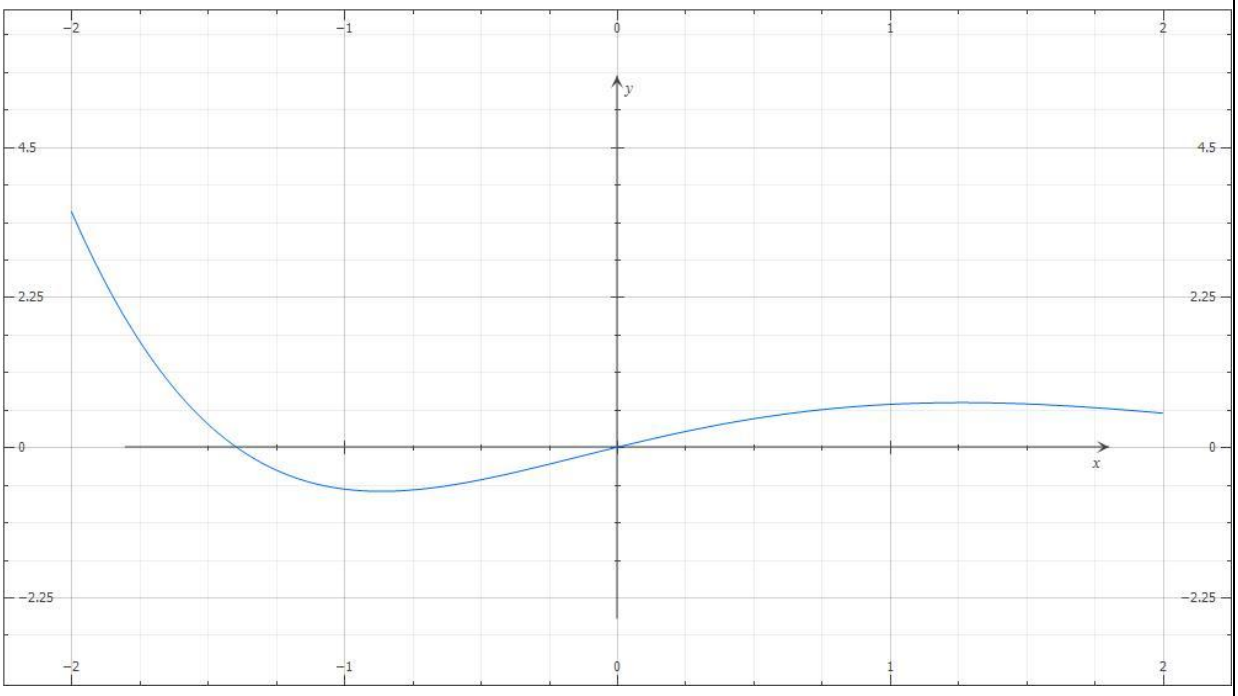
```
{
for(r=0;r<w;r++)
cout<<B[e][r]<<" ";
cout<<"\n";
}


if(del_not<del_tol)
cout<<"Enter scale parameter which is larger than tolerance:";
else
{
while(del_not>del_tol)
{
int flag=0;
for(int e=0;e<2*w;e++)
{
for(int r=0;r<w;r++)

y[r]=x[r]+B[e][r]*del_not;
if(f(y,w)<f(x,w)-pow(del_not,2))
{
for(int r=0;r<w;r++)
x[e]=y[r];
del_not=2*del_not;
flag=1;
break;
}


}
if(flag==0)
del_not/=2;
}
}
```

```
cout<<x[0]<<"\t"<<x[1]<<"\t"<<x[2]<<"\t"<<x[3]<<endl;
cout<<"The minimum value is "<<f(x,d)<<endl;
cout<<"The step length value is "<<del_not<<endl;
return 0;
}
```

OUTPUT

```
Enter number of independent variables:  1
Enter initialassumption: 1

Enter scale control parameter <del-not>: .1
Enter matrixelements:  2
 3
The minimum value is -0.59193323
The step length value is 8.67362e-20
Compilation time: 6.41 sec
```

Graph:



CHAPTER 3

ANALYSIS

8. Comparison of output with original values:

a) f(x) =e^x

It has no minimum value. However, its value tends to zero as x tends to infinity. Using this method, the value iterates to minus 7thpower of 10 for a step length of the value of minus 20th power of 10. Under higher efficiency, the value will tend more towards zero.

b) f(x) = $|x|$

Graphically, its minimum value is zero at x = 0. Although it is not derivable at zero and hence derivative methods are ineffective to find its minima. Using this method, the value iterates to minus 23rd power of 10 for step length of minus 20th power of 10.

c) f(x) = x$sin\frac{1}{x}$

The minimum value for this function comes out to be -0.217233628211. If f(x) is differentiated, the minima comes for a point such that tan(1/x) = (1/x), which gives x = 4.49340945790906. The output of the computing using derivative free method is -0.217913252 and the step length value is 6.93862e-20.

d) f(x) = $\frac{\sin x}{x}$ - $e^{-x}\cos x$

Graphically, the value of the minima of this function is approximately -0.6. Using the direct search method, -0.59193323 comes out to be its minimum value. The step length value is 8.67362e-20.

We observe that the error compared to the actual value is minimal. This error can still be reduced if we reduce the tolerance level of the iteration.

9. Complexity Analysis:

The compilation time varies from 2-3 seconds for f(x) = $|x|$ and f(x) = x$sin\frac{1}{x}$ and about 6.5 seconds for f(x) =e^x and f(x) = $\frac{\sin x}{x}$ - e^{-x} $\cos x$which shows low time complexity. Also, compared to other methods, this method compiles in lesser time with higher degree of polynomial. Since the method does not depend on differentiation, functions not differentiable cannot create issue with time complexity. It is a modern method and it highly efficient for computation compared to other traditional methods.

Conclusion

The two derivative free methods, namely Direct Search Method and Trust Region Method and comparatively new methods which deal with optimization of functions without having to differentiate them. It deals with iterative methods to find absolute optimum value. It is highly efficient to computational techniques and has low time complexity. In fact, the complexity is lower than the derivative methods computation when used for polynomial of higher order. Finally, it's a more general method considering the fact that it deals with all kinds of function, ones which are derivable and the ones which are not or have issues with differentiation, as mentioned in the various cases we dealt with during the working of project.

References

1] S. J. W. Jorge Nocedal, "Numerical Optimization," *Springer* , pp. 34-97, 1999.

2] L. Vicente, "Analysis of Direct Searhes for Discontinious Function," *Mathematical Programming*, pp. 299-325, 12/02/2010.

3] M. P. a. P. L. T. Updating the regularization parameter in the ad N. I. M. Gould, "Updating the regularization parameter in the adaptive cubic regularization algorithm," *Report naXys*, 2011.

4] N. I. M. G. a. P. L. T.] On the complexity of the steepest-descent method with eC. Cartis, "line searching," *Report NAXYS*, 2012.

5] M. J., D. Davies and W. H. Swann, "Non-Linear optimisation Techniques," *Oliver & Boyd.*,
1969.

6] N. I. M. G. P. L. T. Andrew R. Conn, "Trust-Region Methods," *SIAM*.

7] R. H. R. B. S. a. G. A. S. Byrd, "A trust region algorithm for nonlinearly constrained optimization," *SIAM J. Numer. Anal.*, p. 1152–1170, 1987.

8] D. C. Sorensen, "Newton's Method with a Model Trust Region Modification," *SIAM J. Numer. Anal.*, pp. 409-426, 1982.

9] Y. Yuan, "A review of trust region algorithms for optimization" in ICIAM 99:, Edinburgh: Proceedings of the Fourth International Congress on Industrial & Applied Mathematics, Edinburgh, 2000 Oxford University Press, USA, 2000.

10] Y. Yuan, ""Recent Advances in Trust Region Algorithms"," *Math. Program*, 2015.